# 1 Introduction

Linear Matrix Inequalities (LMIs) and LMI techniques have emerged as powerful design tools in areas ranging from control engineering to system identification and structural design. Three factors make LMI techniques appealing:

- a variety of design specifications and constraints can be expressed as LMIs
- once formulated in terms of LMIs, a problem can be solved *exactly* by efficient convex optimization algorithms (the "LMI solvers")
- while most problems with multiple constraints or objectives lack analytical solutions in terms of matrix equations, they often remain tractable in the LMI framework. This makes LMI-based design a valuable alternative to classical "analytical" methods.

See [9] for a good introduction to LMI concepts.

The LMI Control Toolbox is designed as an easy and progressive gateway to the new and fast-growing field of LMIs:

- for users mainly interested in applying LMI techniques to control design, the LMI Control Toolbox features a variety of high-level tools for the analysis and design of multivariable feedback loops (see Chapters 2 through 7)
- for users who occasionally need to solve LMI problems, the "LMI Editor" and the tutorial introduction to LMI concepts and LMI solvers provide for quick and easy problem solving
- for more experienced LMI users, the "LMI Lab" (Chapter 8) offers a rich, flexible, and fully programmable environment to develop customized LMI-based tools.

The LMI Control Toolbox implements state-of-the-art interior-point LMI solvers. While these solvers are significantly faster than classical convex optimization algorithms, it should be kept in mind that the complexity of LMI computations remains higher than that of solving, say, a Riccati equation. For instance, problems with a thousand design variables typically take over an hour on today's workstations. However, research on LMI optimization is still

very active and substantial speed-ups can be expected in the future. Thanks to its efficient "structured" representation of LMIs, the LMI Control Toolbox is geared to making the most out of such improvements.

# Toolbox Features

The LMI Control Toolbox serves two purposes:

- provide state-of-the-art tools for the LMI-based analysis and design of robust control systems
- offer a flexible and user-friendly environment to specify and solve general LMI problems (the **LMI Lab**).

The control design tools can be used without *a priori* knowledge about LMIs or LMI solvers. These are dedicated tools covering the following applications of LMI techniques:

- specification and manipulation of **uncertain dynamical systems** (linear-time invariant, polytopic, parameter-dependent, etc.)
- **robustness analysis**. Various measures of robust stability and performance are implemented, including quadratic stability, techniques based on parameter-dependent Lyapunov functions, $\mu$ analysis, and Popov analysis
- multi-model/multi-objective **state-feedback design**
- **synthesis of output-feedback $H_\infty$ controllers** via Riccati- and LMI-based techniques, including mixed $H_2/H_\infty$ synthesis with regional pole placement constraints
- **loop-shaping** design
- synthesis of **robust gain-scheduled controllers** for time-varying parameter-dependent systems.

For users interested in developing their own applications, the **LMI Lab** provides a general-purpose and fully programmable environment to specify and solve virtually any LMI problem. Note that the scope of this facility is by no means restricted to control-oriented applications.

# LMIs and LMI Problems

A linear matrix inequality (LMI) is any constraint of the form

$$A(x) := A_0 + x_1 A_1 + \cdots + x_N A_N < 0 \tag{1.1}$$

where

- $x = (x_1, \ldots, x_N)$ is a vector of unknown scalars (the *decision* or *optimization* variables)
- $A_0, \ldots, A_N$ are given *symmetric* matrices
- "$< 0$" stands for "negative definite", i.e., the largest eigenvalue of $A(x)$ is negative.

Note that the constraints $A(x) > 0$ and $A(x) < B(x)$ are special cases of (1.1) since they can be rewritten as $-A(x) < 0$ and $A(x) - B(x) < 0$, respectively.

The LMI (1.1) is a convex constraint on $x$ since $A(y) < 0$ and $A(z) < 0$ imply that $A(\frac{y+z}{2}) < 0$. As a result,

- its solution set, called the *feasible set*, is a convex subset of $\mathbf{R}^N$
- finding a solution $x$ to (1.1), if any, is a convex optimization problem.

Convexity has an important consequence: even though (1.1) has no analytical solution in general, it can be solved numerically with guarantees of finding a solution when one exists. Note that a system of LMI constraints can be regarded as a single LMI since

$$\begin{cases} A_1(x) < 0 \\ \quad \vdots \\ A_K(x) < 0 \end{cases} \quad \text{is equivalent to} \quad A(x) := \mathrm{diag}(A_1(x), \ldots, A_K(x)) < 0$$

where $\mathrm{diag}(A_1(x), \ldots, A_K(x))$ denotes the block-diagonal matrix with $A_1(x), \ldots, A_K(x)$ on its diagonal. Hence multiple LMI constraints can be imposed on the vector of decision variables $x$ without destroying convexity.

In most control applications, LMIs do not naturally arise in the canonical form (1.1), but rather in the form

$$\mathcal{L}(X_1, \ldots, X_n) < \mathcal{R}(X_1, \ldots, X_n)$$

where $\mathcal{L}(\cdot)$ and $\mathcal{R}(\cdot)$ are affine functions of some structured *matrix* variables $X_1, \ldots, X_n$. A simple example is the Lyapunov inequality

$$A^T X + XA < 0 \tag{1.2}$$

where the unknown $X$ is a symmetric matrix. Defining $x_1, \ldots, x_N$ as the independent scalar entries of $X$, this LMI could be rewritten in the form (1.1). Yet it is more convenient and efficient to describe it in its natural form (1.2), which is the approach taken in the LMI Lab.

## The Three Generic LMI Problems

Finding a solution $x$ to the LMI system

$$A(x) < 0 \tag{1.3}$$

is called the *feasibility problem*. Minimizing a convex objective under LMI constraints is also a convex problem. In particular, the *linear objective minimization problem*

$$\text{Minimize } c^T x \text{ subject to } A(x) < 0 \tag{1.4}$$

plays an important role in LMI-based design. Finally, the *generalized eigenvalue minimization problem*

$$\text{Minimize } \lambda \text{ subject to } \begin{cases} A(x) < \lambda B(x) \\ B(x) > 0 \\ C(x) < 0 \end{cases} \tag{1.5}$$

is quasi-convex and can be solved by similar techniques. It owes its name to the fact that $\lambda$ is related to the largest generalized eigenvalue of the pencil $(A(x), B(x))$.

# LMIs in Control

Many control problems and design specifications have LMI formulations [9]. This is especially true for Lyapunov-based analysis and design, but also for optimal LQG control, $H_\infty$ control, covariance control, etc. Further applications of LMIs arise in estimation, identification, optimal design, structural design [6, 7], matrix scaling problems, and so on. The main strength of LMI formulations is the ability to combine various design constraints or objectives in a numerically tractable manner.

A non-exhaustive list of problems addressed by LMI techniques includes:

- robust stability of systems with LTI uncertainty ($\mu$-analysis) [24, 21, 27]
- robust stability in the face of sector-bounded nonlinearities (Popov criterion) [22, 28, 13, 16]

- quadratic stability of differential inclusions [15, 8]
- Lyapunov stability of parameter-dependent systems [12]
- input/state/output properties of LTI systems (invariant ellipsoids, decay rate, etc.) [9]
- multi-model/multi-objective state feedback design [4, 17, 3, 9, 10]
- robust pole placement
- optimal LQG control [9]
- robust $H_\infty$ control [11, 14]
- multi-objective $H_\infty$ synthesis [17, 23, 10, 18]
- design of robust gain-scheduled controllers [5, 2]
- control of stochastic systems [9]
- weighted interpolation problems [9].

To hint at the principles underlying LMI design, we review the LMI formulations of a few typical design objectives:

**Stability:** the stability of the dynamical system

$$\dot{x} = A\,x$$

is equivalent to the feasibility of

$$\text{Find } P = P^T \text{ such that } \; A^T P + PA < 0\,, \quad P > I.$$

This can be generalized to linear differential inclusions (LDI)

$$\dot{x} = A(t)\,x$$

where $A(t)$ varies in the convex envelope of a set of LTI models:

$$A(t) \in \mathrm{Co}\{A_1, \ldots, A_n\} = \left\{ \sum_{i=1}^{n} \alpha_i\, A_i \;\; : \;\; \alpha_i \geq 0, \;\; \sum_{i=1}^{N} \alpha_i = 1 \right\}.$$

A sufficient condition for the asymptotic stability of this LDI is the feasibility of

$$\text{Find } P = P^T \text{ such that } \; A_i^T P + PA_i < 0\,, \quad P > I.$$

**RMS gain:** the random-mean-squares (RMS) gain of a stable LTI system

$$\begin{cases} \dot{x} = A\,x + B\,u \\ y = C\,x + D\,u \end{cases}$$

is the largest input/output gain over all bounded inputs $u(t)$. This gain is the global minimum of the following linear objective minimization problem [1, 26, 25]:

Minimize $\gamma$ over $X = X^T$ and $\gamma$ such that

$$\begin{pmatrix} A^TX + XA & XB & C^T \\ B^TX & -\gamma I & D^T \\ C & D & -\gamma I \end{pmatrix} < 0$$

$$X > 0.$$

**LQG performance:** for a stable LTI system

$$G \begin{cases} \dot{x} = A\,x + B\,w \\ y = C\,x \end{cases}$$

where $w$ is a white noise disturbance with unit covariance, the LQG or $H_2$ performance $\|G\|_2$ is defined by

$$\begin{aligned} \|G\|_2^2 &:= \lim_{T\to\infty} E\left\{\frac{1}{T}\int_0^T y^T(t)y(t)dt\right\} \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} G^H(j\omega)\,G(j\omega)\,d\omega \end{aligned}$$

It can be shown that

$$\|G\|_2^2 = \inf\left\{\text{Trace}\,(CPC^T) \;:\; AP + PA^T + BB^T < 0\right\}$$

Hence $\|G\|_2^2$ is the global minimum of the LMI problem

Minimize Trace $(Q)$ over the symmetric matrices $P, Q$ such that

$$\begin{aligned} AP + PA^T + BB^T &< 0 \\ \begin{pmatrix} Q & CP \\ PC^T & P \end{pmatrix} &> 0. \end{aligned}$$

Again this is a linear objective minimization problem since the objective Trace$(Q)$ is linear in the decision variables (free entries of $P, Q$).

# Further Mathematical Background

Efficient interior-point algorithms are now available to solve the three generic LMI problems (1.3)-(1.5) defined on p. 1-4. These algorithms have a polynomial-time complexity. That is, the number $N(\varepsilon)$ of flops needed to compute an $\varepsilon$-accurate solution is bounded by

$$N(\varepsilon) \leq M\, N^3\, \log(V/\varepsilon)$$

where $M$ is the total row size of the LMI system, $N$ is the total number of scalar decision variables, and $V$ is a data-dependent scaling factor. The LMI Control Toolbox implements the Projective Algorithm of Nesterov and Nemirovski [20, 19]. In addition to its polynomial-time complexity, this algorithm does not require an initial feasible point for the linear objective minimization problem (1.4) or the generalized eigenvalue minimization problem (1.5).

Some LMI problems are formulated in terms of inequalities rather than strict inequalities. For instance, a variant of (1.4) is

$$\text{Minimize } c^T x \text{ subject to } A(x) \leq 0 .$$

While this distinction is immaterial in general, it matters when $A(x)$ can be made negative semi-definite but not negative definite. A simple example is

$$\text{Minimize } c^T x \text{ subject to } \begin{pmatrix} x & x \\ x & x \end{pmatrix} \geq 0 . \tag{1.6}$$

Such problems cannot be handled directly by interior-point methods which require strict feasibility of the LMI constraints. A well-posed reformulation of (1.6) would be

$$\text{Minimize } c^T x \text{ subject to } x \geq 0 .$$

Keeping this subtlety in mind, we always use strict inequalities in this manual.

# References


[1] Anderson, B.D.O, and S. Vongpanitlerd, *Network Analysis*, Prentice Hall, Englewood Cliffs, 1973.

[2] Apkarian, P., P. Gahinet, and G. Becker, "Self-Scheduled $H_\infty$ Control of Linear Parameter-Varying Systems," *Proc. Amer. Contr. Conf.*, 1994, pp. 856-860.

[3] Bambang, R., E. Shimemura, and K. Uchida, "Mixed $H_2/H_\infty$ Control with Pole Placement," State-Feedback Case," *Proc. Amer. Contr. Conf.*, 1993, pp. 2777-2779.

[4] Barmish, B.R., "Stabilization of Uncertain Systems via Linear Control," *IEEE Trans. Aut. Contr.*, AC-28 (1983), pp. 848-850.

[5] Becker, G., Packard, P., "Robust Performance of Linear-Parametrically Varying Systems Using Parametrically-Dependent Linear Feedback," *Systems and Control Letters*, 23 (1994), pp. 205-215.

[6] Bendsoe, M.P., A. Ben-Tal, and J. Zowe, "Optimization Methods for Truss Geometry and Topology Design," to appear in *Structural Optimization.*

[7] Ben-Tal, A., and A. Nemirovski, "Potential Reduction Polynomial-Time Method for Truss Topology Design," to appear in *SIAM J. Contr. Opt.*.

[8] Boyd, S., and Q. Yang, "Structured and Simultaneous Lyapunov Functions for System Stability Problems," *Int. J. Contr.*, 49 (1989), pp. 2215-2240.

[9] Boyd, S., L. El Ghaoui, E. Feron, V. Balakrishnan, *Linear Matrix Inequalities in Systems and Control Theory*, SIAM books, Philadelphia, 1994.

[10] Chilali, M., and P. Gahinet, "$H_\infty$ Design with Pole Placement Constraints: an LMI Approach," to appear in *IEEE Trans. Aut. Contr.*. Also in *Proc. Conf. Dec. Contr.*, 1994, pp. 553-558.

[11] Gahinet, P., and P. Apkarian, "A Linear Matrix Inequality Approach to $H_\infty$ Control," *Int. J. Robust and Nonlinear Contr.*, 4 (1994), pp. 421-448.

[12] Gahinet, P., P. Apkarian, and M. Chilali, "Affine Parameter-Dependent Lyapunov Functions for Real Parametric Uncertainty," *Proc. Conf. Dec. Contr.*, 1994, pp. 2026-2031.

[13] Haddad, W.M. and D.S. Berstein,"Parameter-Dependent Lyapunov Functions, Constant Real Parameter Uncertainty, and the Popov Criterion in Robust Analysis and Synthesis: Part 1 and 2," *Proc. Conf. Dec. Contr.*, 1991, pp. 2274-2279 and 2632-2633.

[14] Iwasaki, T., and R.E. Skelton, "All Controllers for the General $H_\infty$ Control Problem: LMI Existence Conditions and State-Space Formulas," *Automatica*, 30 (1994), pp. 1307-1317.

[15] Horisberger, H.P., and P.R. Belanger, "Regulators for Linear Time-Varying Plants with Uncertain Parameters," *IEEE Trans. Aut. Contr.*, AC-21 (1976), pp. 705-708.

[16] How, J.P., and S.R. Hall, "Connection between the Popov Stability Criterion and Bounds for Real Parameter Uncertainty," *Proc. Amer. Contr. Conf.*, 1993, pp. 1084-1089.

[17] Khargonekar, P.P., and M.A. Rotea,"Mixed $H_2/H_\infty$ Control: a Convex Optimization Approach," *IEEE Trans. Aut. Contr.*, 39 (1991), pp. 824-837.

[18] Masubuchi, I., A. Ohara, and N. Suda, "LMI-Based Controller Synthesis: A Unified Formulation and Solution," submitted to *Int. J. Robust and Nonlinear Contr.*, 1994.

[19] Nemirovski, A., and P. Gahinet, "The Projective Method for Solving Linear Matrix Inequalities," *Proc. Amer. Contr. Conf.*, 1994, pp. 840-844.

[20] Nesterov, Yu, and A. Nemirovski, *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, SIAM Books, Philadelphia, 1994.

[21] Packard, A., and J.C. Doyle, "The Complex Structured Singular Value," *Automatica*, 29 (1994), pp. 71-109.

[22] Popov, V.M., "Absolute Stability of Nonlinear Systems of Automatic Control," *Automation and Remote Control*, 22 (1962), pp. 857-875.

[23] Scherer, C., "Mixed $H_2 H_\infty$ Control," to appear in *Trends in Control: A European Perspective*, volume of the special contributions to the ECC 1995.

[24] Stein, G. and J.C. Doyle, "Beyond Singular Values and Loop Shapes," *J. Guidance*, 14 (1991), pp. 5-16.

[25] Vidyasagar, M., *Nonlinear System Analysis*, Prentice-Hall, Englewood Cliffs, 1992.

[26] Willems, J.C., "Least-squares Stationary Optimal Control and the Algebraic Riccati Equation," *IEEE Trans. Aut. Contr.*, AC-16 (1971), pp. 621-634.

[27] Young, P. M., M. P. Newlin, and J. C. Doyle, "Let's Get Real," in *Robust Control Theory*, Springer Verlag, 1994, pp. 143-174.

[28] Zames, G., "On the Input-Output Stability of Time-Varying Nonlinear Feedback Systems, Part I and II," *IEEE Trans. Aut. Contr.*, AC-11 (1966), pp. 228-238 and 465-476.

# 8 The LMI Lab

The LMI Lab is a high-performance package for solving general LMI problems. It blends simple tools for the specification and manipulation of LMIs with powerful LMI solvers for three generic LMI problems. Thanks to a structure-oriented representation of LMIs, the various LMI constraints can be described in their natural block-matrix form. Similarly, the optimization variables are specified directly as *matrix variables* with some given structure. Once an LMI problem is specified, it can be solved numerically by calling the appropriate LMI solver. The three solvers `feasp`, `mincx`, and `gevp` constitute the computational engine of the LMI Control Toolbox. Their high performance is achieved through C-MEX implementation and by taking advantage of the particular structure of each LMI.

The LMI Lab offers tools to

- specify LMI systems either symbolically with the LMI Editor or incrementally with the `lmivar` and `lmiterm` commands
- retrieve information about existing systems of LMIs
- modify existing systems of LMIs
- solve the three generic LMI problems (feasibility problem, linear objective minimization, and generalized eigenvalue minimization)
- validate results

This chapter gives a tutorial introduction to the LMI Lab as well as more advanced tips for making the most out of its potential. The tutorial material is covered by the demo `lmidem`.

# Background and Terminology

Any linear matrix inequality can be expressed in the *canonical form*

$$L(x) = L_0 + x_1 L_1 + \ldots + x_N L_N < 0$$

where

- $L_0, L_1, \ldots, L_N$ are given symmetric matrices
- $x = (x_1, \ldots, x_N)^T \in \mathbf{R}^N$ is the vector of scalar variables to be determined. We refer to $x_1, \ldots, x_N$ as the *decision variables*. The names "design variables" and "optimization variables" are also found in the literature.

Even though this canonical expression is generic, LMIs rarely arise in this form in control applications. Consider for instance the Lyapunov inequality

$$A^T X + XA < 0 \tag{8.1}$$

where $A = \begin{pmatrix} -1 & 2 \\ 0 & -2 \end{pmatrix}$ and the variable $X = \begin{pmatrix} x_1 & x_2 \\ x_2 & x_3 \end{pmatrix}$ is a symmetric matrix. Here the decision variables are the free entries $x_1, x_2, x_3$ of $X$ and the canonical form of this LMI reads

$$x_1 \begin{pmatrix} -2 & 2 \\ 2 & 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 & -3 \\ -3 & 4 \end{pmatrix} + x_3 \begin{pmatrix} 0 & 0 \\ 0 & -4 \end{pmatrix} < 0. \tag{8.2}$$

Clearly this expression is less intuitive and transparent than (8.1). Moreover, the number of matrices involved in (8.2) grows roughly as $n^2/2$ if $n$ is the size of the $A$ matrix. Hence, the canonical form is very inefficient from a storage viewpoint since it requires storing $o(n^2/2)$ matrices of size $n$ when the single $n$-by-$n$ matrix $A$ would be sufficient. Finally, working with the canonical form is also detrimental to the efficiency of the LMI solvers. For these various reasons, the LMI Lab uses a **structured representation** of LMIs. For instance, the expression $A^T X + XA$ in the Lyapunov inequality (8.1) is explicitly described as a function of the matrix variable $X$, and only the $A$ matrix is stored.

In general, LMIs assume a block matrix form where each block is an affine combination of the matrix variables. As a fairly typical illustration, consider the following LMI drawn from $H_\infty$ theory

$$\mathcal{N}^T \begin{pmatrix} A^T X + XA & XC^T & B \\ CX & -\gamma I & D \\ B^T & D^T & -\gamma I \end{pmatrix} \mathcal{N} < 0 \tag{8.3}$$

where $A, B, C, D, \mathcal{N}$ are given matrices and the problem variables are $X = X^T \in \mathbf{R}^{n\times n}$ and $\gamma \in \mathbf{R}$. We use the following terminology to describe such LMIs:

- $\mathcal{N}$ is called the *outer factor*, and the block matrix

$$L(X,\gamma) = \begin{pmatrix} A^TX + XA & XC^T & B \\ CX & -\gamma I & D \\ B^T & D^T & -\gamma I \end{pmatrix}$$

  is called the *inner factor*. The outer factor **needs not be square** and is **often absent.**

- $X$ and $\gamma$ are the *matrix variables* of the problem. Note that scalars are considered as $1 \times 1$ matrices.
- The inner factor $L(X,\gamma)$ is a symmetric *block matrix*, its block structure being characterized by the sizes of its diagonal blocks. By symmetry, $L(X,\gamma)$ is entirely specified by the blocks on or above the diagonal.
- Each block of $L(X,\gamma)$ is an affine expression in the matrix variables $X$ and $\gamma$. This expression can be broken down into a sum of elementary *terms*. For instance, the block (1,1) contains two elementary terms: $A^TX$ and $XA$.

  Terms are either *constant* or *variable*. Constant terms are fixed matrices like $B$ and $D$ above. Variable terms involve one of the matrix variables, like $XA$, $XC^T$, and $-\gamma I$ above.

The LMI (8.3) is fully specified by the list of terms in each block, as is any LMI regardless of its complexity.

As for the matrix variables $X$ and $\gamma$, they are characterized by their dimensions and structure. Common structures include rectangular unstructured, symmetric, skew-symmetric, and scalar. More sophisticated structures are sometimes encountered in control problems. For instance, the matrix variable $X$ could be constrained to the block-diagonal structure

$$X = \left(\begin{array}{c|cc} x_1 & 0 & 0 \\ \hline 0 & x_2 & x_3 \\ 0 & x_3 & x_4 \end{array}\right).$$

Another possibility is the symmetric Toeplitz structure

$$X = \begin{pmatrix} x_1 & x_2 & x_3 \\ x_2 & x_1 & x_2 \\ x_3 & x_2 & x_1 \end{pmatrix}.$$

Summing up, structured LMI problems are specified by declaring the matrix variables and describing the term content of each LMI. This term-oriented description is systematic and accurately reflects the specific structure of the LMI constraints. There is no built-in limitation on the number of LMIs that can be specified or on the number of blocks and terms in any given LMI. Hence LMI systems of arbitrary complexity can be defined in the LMI Lab.

# Overview of the LMI Lab

The LMI Lab offers tools to specify, manipulate, and numerically solve LMIs. Its main purpose is to

- allow for straightforward description of LMIs in their natural block-matrix form
- provide easy access to the LMI solvers (optimization codes)
- facilitate result validation and problem modification.

The structure-oriented description of a given LMI system is stored as a single vector called the *internal representation* and generically denoted by `LMISYS` in the sequel. This vector encodes the structure and dimensions of the LMIs and matrix variables, a description of all LMI terms, and the related numerical data. It must be stressed that users need not attempt to read or understand the content of `LMISYS` since all manipulations involving this internal representation can be performed in a transparent manner with LMI-Lab tools.

The LMI Lab supports the following functionalities:

**Specification of a system of LMIs**
LMI systems can be either specified as symbolic matrix expressions with the interactive graphical user interface `lmiedit`, or assembled incrementally with the two commands `lmivar` and `lmiterm`. The first option is more intuitive and transparent while the second option is more powerful and flexible.

**Information retrieval**
The interactive function `lmiinfo` answers qualitative queries about LMI systems created with `lmiedit` or `lmivar` and `lmiterm`. You can also use `lmiedit` to visualize the LMI system produced by a particular sequence of `lmivar`/`lmiterm` commands.

**Solvers for LMI optimization problems**
General-purpose LMI solvers are provided for the three generic LMI problems defined on p. 1-4. These solvers can handle very general LMI systems and

matrix variable structures. They return a feasible or optimal vector of decision variables $x^*$. The corresponding values $X_1^*, \ldots, X_K^*$ of the matrix variables are given by the function `dec2mat`.

**Result validation**
The solution $x^*$ produced by the LMI solvers is easily validated with the functions `evallmi` and `showlmi`. This allows a fast check and/or analysis of the results. With `evallmi`, all variable terms in the LMI system are evaluated for the value $x^*$ of the decision variables. The left- and right-hand sides of each LMI then become constant matrices that can be displayed with `showlmi`.

**Modification of a system of LMIs**
An existing system of LMIs can be modified in two ways:

- An LMI can be removed from the system with `dellmi`.
- A matrix variable $X$ can be deleted using `delmvar`. It can also be instantiated, that is, set to some given matrix value. This operation is performed by `setmvar` and allows, for example, to fix some variables and solve the LMI problem with respect to the remaining ones.

# Specifying a System of LMIs

The LMI Lab can handle any system of LMIs of the form

$$\mathcal{N}^T \ \mathcal{L}(X_1, \ldots, X_K) \ \mathcal{N} \ < \ \mathcal{M}^T \ \mathcal{R}(X_1, \ldots, X_K) \ \mathcal{M}$$

where

- $X_1, \ldots, X_K$ are matrix variables with some prescribed structure
- the left and right outer factors $\mathcal{N}$ and $\mathcal{M}$ are given matrices with *identical* dimensions
- the left and right inner factors $\mathcal{L}(\cdot)$ and $\mathcal{R}(\cdot)$ are symmetric block matrices with *identical* block structures, each block being an affine combination of $X_1, \ldots, X_K$ and their transposes.

> **IMPORTANT:** Throughout this chapter, "left-hand side" refers to what is on the "smaller" side of the inequality, and "right-hand side" to what is on the "larger" side. Accordingly, $X$ is called the right-hand side and 0 the left-hand side of the LMI
>
> $$0 < X$$
>
> even when this LMI is written as
>
> $$X > 0 .$$

The specification of an LMI system involves two steps:

1. Declare the dimensions and structure of each matrix variable $X_1, \ldots, X_K$.
2. Describe the term content of each LMI.

This process creates the so-called *internal representation* of the LMI system. This computer description of the problem is used by the LMI solvers and in all subsequent manipulations of the LMI system. It is stored as a single vector which we consistently denote by `LMISYS` in the sequel (for the sake of clarity).

There are two ways of generating the internal description of a given LMI system: (1) by a sequence of `lmivar`/`lmiterm` commands that build it incrementally, or (2) via the LMI Editor `lmiedit` where LMIs can be specified directly as symbolic matrix expressions. Though somewhat less flexible and powerful than the command-based description, the LMI Editor is more straightforward to use, hence particularly well-suited for beginners. Thanks to its coding and decoding capabilities, it also constitutes a good tutorial introduction to `lmivar` and `lmiterm`. Accordingly, beginners may elect to skip the subsections on `lmivar` and `lmiterm` and to concentrate on the GUI-based specification of LMIs with `lmiedit`.

## A Simple Example

The following tutorial example is used to illustrate the specification of LMI systems with the LMI-Lab tools. Run the demo `lmidem` to see a complete treatment of this example.

**Example 8.1:** Consider a stable transfer function

$$G(s) = C(sI - A)^{-1}B \tag{8.4}$$

with 4 inputs, 4 outputs, and 6 states, and consider the set of input/output scaling matrices $D$ with block-diagonal structure

$$D = \begin{pmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_1 & 0 & 0 \\ 0 & 0 & d_2 & d_3 \\ 0 & 0 & d_4 & d_5 \end{pmatrix}. \tag{8.5}$$

The following problem arises in the robust stability analysis of systems with time-varying uncertainty [4]:

> Find, if any, a scaling $D$ of structure (8.5) such that the largest gain across frequency of $D\,G(s)\,D^{-1}$ is less than one.

This problem has a simple LMI formulation: there exists an adequate scaling $D$ iff. the following feasibility problem has solutions:

> Find two symmetric matrices $X \in \mathbf{R}^{6\times 6}$ and $S = D^T D \in \mathbf{R}^{4\times 4}$ such that

$$\begin{pmatrix} A^T X + XA + C^T SC & XB \\ B^T X & -S \end{pmatrix} < 0 \tag{8.6}$$

$$X > 0 \tag{8.7}$$

$$S > I. \tag{8.8}$$

The LMI system (8.6)-(8.8) can be described with the LMI Editor as outlined on p. 8-13 below. Alternatively, its internal description can be generated with `lmivar` and `lmiterm` commands as follows:

```
setlmis([])
X=lmivar(1,[6 1])
S=lmivar(1,[2 0;2 1])

% 1st LMI
lmiterm([1 1 1 X],1,A,'s')
lmiterm([1 1 1 S],C',C)
lmiterm([1 1 2 X],1,B)
lmiterm([1 2 2 S],-1,1)

% 2nd LMI
lmiterm([-2 1 1 X],1,1)

% 3rd LMI
lmiterm([-3 1 1 S],1,1)
lmiterm([3 1 1 0],1)

LMISYS' = getlmis
```

Here the `lmivar` commands define the two matrix variables $X$ and $S$ while the `lmiterm` commands describe the various terms in each LMI. Upon completion, `getlmis` returns the internal representation `LMISYS` of this LMI system. The next three subsections give more details on the syntax and usage of these various commands. More information on how the internal representation is updated by `lmivar/lmiterm` can also be found on p. 8-15.

## setlmis and getlmis

The description of an LMI system should begin with `setlmis` and end with `getlmis`. The function `setlmis` initializes the LMI system description. When specifying a new system, type

```
setlmis([])
```

To add on to an existing LMI system with internal representation `LMIS0`, type

```
setlmis(LMIS0)
```

When the LMI system is completely specified, type

```
LMISYS = getlmis
```

This returns the internal representation `LMISYS` of this LMI system. ThisMATLAB description of the problem can be forwarded to other LMI-Lab functions for subsequent processing. The command `getlmis` must be used **only once** and after declaring **all** matrix variables and LMI terms.

## lmivar

The matrix variables are declared one at a time with `lmivar` and are characterized by their structure. To facilitate the specification of this structure, the LMI Lab offers two predefined structure types along with the means to describe more general structures:

Type 1: **Symmetric block diagonal** structure. This corresponds to matrix variables of the form

$$X = \begin{pmatrix} D_1 & 0 & \cdots & 0 \\ 0 & D_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & D_r \end{pmatrix}$$

where each diagonal block $D_j$ is square and is either zero, a *full symmetric* matrix, or a *scalar matrix*

$$D_j = d \times I\,, \qquad d \in \mathbf{R}.$$

This type encompasses ordinary symmetric matrices (single block) and scalar variables (one block of size one).

Type 2: **Rectangular** structure. This corresponds to arbitrary rectangular matrices without any particular structure.

Type 3: **General** structures. This third type is used to describe more sophisticated structures and/or correlations between the matrix variables. The principle is as follows: each entry of $X$ is specified independently as either 0, $x_n$, or $-x_n$ where $x_n$ denotes the $n$-th decision variable in the problem. For details on how to use Type 3, see pp. 8-25 through 8-28 below as well as the `lmivar` entry of the *Command Reference* chapter.

In Example 8.1, the matrix variables $X$ and $S$ are of Type 1. Indeed, both are symmetric and $S$ inherits the block-diagonal structure (8.5) of $D$. Specifically, $S$ is of the form

$$S = \begin{pmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_1 & 0 & 0 \\ 0 & 0 & s_2 & s_3 \\ 0 & 0 & s_3 & s_4 \end{pmatrix}.$$

After initializing the description with the command `setlmis([])`, these two matrix variables are declared by

```
lmivar(1,[6 1])          % X
lmivar(1,[2 0;2 1])      % S
```

In both commands, the first input specifies the structure type and the second input contains additional information about the structure of the variable:

- for a matrix variable $X$ of Type 1, this second input is a matrix with two columns and as many rows as diagonal blocks in $X$. The first column lists the sizes of the diagonal blocks and the second column specifies their nature with the following convention:

  $$\begin{aligned} 1 &\longrightarrow \text{full symmetric block} \\ 0 &\longrightarrow \text{scalar block} \\ -1 &\longrightarrow \text{zero block} \end{aligned}$$

  In the second command, for instance, `[2 0;2 1]` means that $S$ has two diagonal blocks, the first one being a $2 \times 2$ scalar block and the second one a $2 \times 2$ full block.
- for matrix variables of Type 2, the second input of `lmivar` is a two-entry vector listing the row and column dimensions of the variable. For instance, a $3 \times 5$ rectangular matrix variable would be defined by

```
lmivar(2,[3 5])
```

For convenience, `lmivar` also returns a "tag" that identifies the matrix variable for subsequent reference. For instance, $X$ and $S$ in Example 8.1 could be defined by

```
X = lmivar(1,[6 1])
S = lmivar(1,[2 0;2 1])
```

The identifiers `X` and `S` are integers corresponding to the ranking of $X$ and $S$ in the list of matrix variables (in the order of declaration). Here their values would be `X=1` and `S=2`. Note that these identifiers still point to $X$ and $S$ after deletion or instantiation of some of the matrix variables. Finally, `lmivar` can also return the total number of decision variables allocated so far as well as the entry-wise dependence of the matrix variable on these decision variables (see p. 8-25 and the `lmivar` entry of the *Command Reference* chapter for more details).

## lmiterm

After declaring the matrix variables with `lmivar`, we are left with specifying the term content of each LMI. Recall that LMI terms fall into three categories:

- the *constant terms*, i.e., fixed matrices like $I$ in the left-hand side of the LMI $S > I$
- the *variable terms*, i.e., terms involving a matrix variable. For instance, $A^T X$ and $C^T S C$ in (8.6). Variable terms are of the form $PXQ$ where $X$ is a

variable and $P, Q$ are given matrices called the left and right *coefficients*, respectively

- the *outer factors*.

The following rule should be kept in mind when describing the term content of an LMI:

> **IMPORTANT:** specify only the terms in the blocks *on or above* the diagonal. The inner factors being symmetric, this is sufficient to specify the entire LMI. *Specifying all blocks results in the duplication of off-diagonal terms, hence in the creation of a different LMI.* Alternatively, users may describe the blocks on or below the diagonal.

LMI terms are specified one at a time with `lmiterm`. For instance, the LMI

$$\begin{pmatrix} A^TX + XA + C^TSC & XB \\ B^TX & -S \end{pmatrix} < 0$$

is described by

```
lmiterm([1 1 1 1],1,A,'s')
lmiterm([1 1 1 2],C',C)
lmiterm([1 1 2 1],1,B)
lmiterm([1 2 2 2],-1,1)
```

These commands successively declare the terms $A^TX + XA$, $C^TSC$, $XB$, and $-S$. In each command, the first argument is a four-entry vector listing the term characteristics as follows:

- the first entry indicates to which LMI the term belongs. The value `m` means "left-hand side of the $m$-th LMI," and `-m` means "right-hand side of the $m$-th LMI"
- the second and third entries identify the block to which the term belongs. For instance, the vector `[1 1 2 1]` indicates that the term is attached to the $(1,2)$ block
- the last entry indicates which matrix variable is involved in the term. This entry is `0` for constant terms, `k` for terms involving the $k$-th matrix variable $X_k$, and `-k` for terms involving $X_k^T$ (here $X$ and $S$ are first and second variables in the order of declaration).

Finally, the second and third arguments of `lmiterm` contain the numerical data (values of the constant term, outer factor, or matrix coefficients $P$ and $Q$

for variable terms $PXQ$ or $PX^TQ$). These arguments must refer to existing MATLAB variables and be **real-valued**. See p. 8-28 for the specification of LMIs with complex-valued coefficients.

Some shorthand is provided to simplify term specification. First, blocks are zero by default. Secondly, in *diagonal blocks* the extra argument `'s'` allows you to specify the conjugated expression $AXB + B^TX^TA^T$ with a *single* `lmiterm` command. For instance, the first command specifies $A^TX + XA$ as the "symmetrization" of $XA$. Finally, scalar values are allowed as shorthand for *scalar matrices*, i.e., matrices of the form $\alpha I$ with $\alpha$ scalar. Thus, a constant term of the form $\alpha I$ can be specified as the "scalar" $\alpha$. This also applies to the coefficients $P$ and $Q$ of variable terms. The dimensions of scalar matrices are inferred from the context and set to 1 by default. For instance, the third LMI $S > I$ in Example 8.1 is described by

```
lmiterm([-3 1 1 2],1,1)     % 1*S*1 = S
lmiterm([3 1 1 0],1)        % 1*I = I
```

Recall that by convention $S$ is considered as the right-hand side of the inequality, which justifies the $-3$ in the first command.

Finally, to improve readability it is often convenient to attach an identifier (tag) to each LMI and matrix variable. The variable identifiers are returned by `lmivar` and the LMI identifiers are set by the function `newlmi`. These identifiers can be used in `lmiterm` commands to refer to a given LMI or matrix variable. For the LMI system of Example 8.1, this would look like:

```
setlmis([])
X = lmivar(1,[6 1])
S = lmivar(1,[2 0;2 1])

BRL = newlmi
lmiterm([BRL 1 1 X],1,A,'s')
lmiterm([BRL 1 1 S],C',C)
lmiterm([BRL 1 2 X],1,B)
lmiterm([BRL 2 2 S],-1,1)

Xpos = newlmi
lmiterm([-Xpos 1 1 X],1,1)

Slmi = newlmi
lmiterm([-Slmi 1 1 S],1,1)
lmiterm([Slmi 1 1 0],1)

LMISYS = getlmis
```

Here the identifiers `X` and `S` point to the variables $X$ and $S$ while the tags `BRL`, `Xpos`, and `Slmi` point to the first, second, and third LMI, respectively. Note that `-Xpos` refers to the right-hand side of the second LMI. Similarly, `-X` would indicate transposition of the variable $X$.

## The LMI Editor lmiedit

The LMI Editor `lmiedit` is a graphical user interface (GUI) to specify LMI systems in a straightforward symbolic manner. Typing

```
lmiedit
```

calls up a window with several editable text areas and various pushbuttons. To specify your LMI system,

1. Declare each matrix variable (name and structure) in the upper half of the worksheet. The structure is characterized by its type (`S` for symmetric block diagonal, `R` for unstructured, and `G` for other structures) and by an additional "structure" matrix. This matrix contains specific information about the structure and corresponds to the second argument of `lmivar` (see p. 8-8 for details).

   Please use **one line per matrix variable** in the text editing areas

2. Specify the LMIs as MATLAB expressions in the lower half of the worksheet. For instance, the LMI

$$\begin{pmatrix} A^T X + XA & XB \\ B^T X & -I \end{pmatrix} < 0$$

   is entered by typing

```
[a'*x+x*a  x*b ; b'*x  -1] < 0
```

   if `x` is the name given to the matrix variable $X$ in the upper half of the worksheet. The left- and right-hand sides of the LMIs should be *valid* MATLAB expressions.

Once the LMI system is fully specified, the following tasks can be performed by pressing the corresponding button:

- Visualize the sequence of `lmivar`/`lmiterm` commands needed to describe this LMI system (`view commands` buttons). Conversely, the LMI system defined by a particular sequence of `lmivar`/`lmiterm` commands can be displayed as a MATLAB expression by clicking on the `describe...` buttons.

Beginners can use this facility as a tutorial introduction to the `lmivar` and `lmiterm` commands.

- Save the symbolic description of the LMI system as a MATLAB string (`save` button). This description can be reloaded later on by pressing the `load` button.
- Read a sequence of `lmivar`/`lmiterm` commands from a file (`read` button). You can then click on "`describe the matrix variables`" or "`describe the LMIs...`" to visualize the symbolic expression of the LMI system specified by these commands. The file should describe a single LMI system but may otherwise contain any sequence of MATLAB commands.

  This feature is useful for code validation and debugging.
- Write in a file the sequence of `lmivar`/`lmiterm` commands needed to describe a particular LMI system (`write` button).

  This is helpful to develop code and prototype MATLAB functions based on the LMI Lab.
- Generate the internal representation of the LMI system by pressing `create`. The result is written in a MATLAB variable named after the LMI system (if the "name of the LMI system" is set to `mylmi`, the internal representation is written in the MATLAB variable `mylmi`). Note that all LMI-related data should be defined in the MATLAB workspace at this stage.

  The internal representation can be passed directly to the LMI solvers or any other LMI Lab function.

As with `lmiterm`, you can use various shortcuts when entering LMI expressions at the keyboard. For instance, zero blocks can be entered simply as `0` and need not be dimensioned. Similarly, the identity matrix can be entered as `1` without dimensioning. Finally, **upper diagonal** LMI blocks need not be fully specified. Rather, you can just type `(*)` in place of each such block.

Though fairly general, `lmiedit` is not as flexible as `lmiterm` and the following limitations should be kept in mind:

- parentheses cannot be used around matrix variables. For instance, the expression

  ```
  (a*x+b)'*c + c'*(a*x+b)
  ```

  is invalid when `x` is a variable name. By contrast,

  ```
  (a+b)'*x + x'*(a+b)
  ```

  is perfectly valid

- loops and `if` statements are ignored
- when turning `lmiterm` commands into a symbolic description of the LMI system, an error is issued if the first argument of `lmiterm` cannot be evaluated. Use the LMI and variable identifiers supplied by `newlmi` and `lmivar` to avoid such difficulties.

Figure 8-1 shows how to specify the feasibility problem of Example 8.1 on p. 8-6 with `lmiedit`.

## How It All Works

Users familiar with MATLAB may wonder how `lmivar` and `lmiterm` physically update the internal representation `LMISYS` since `LMISYS` is not an argument to these functions. In fact, all updating is performed through global variables for maximum speed. These global variables are initialized by `setlmis`, cleared by `getlmis`, and not visible in the workspace. Even though this artifact is transparent from the user's viewpoint, be sure to

- invoke `getlmis` only once and after completely specifying the LMI system
- refrain from using the command `clear global` before the LMI system description is ended with `getlmis`.

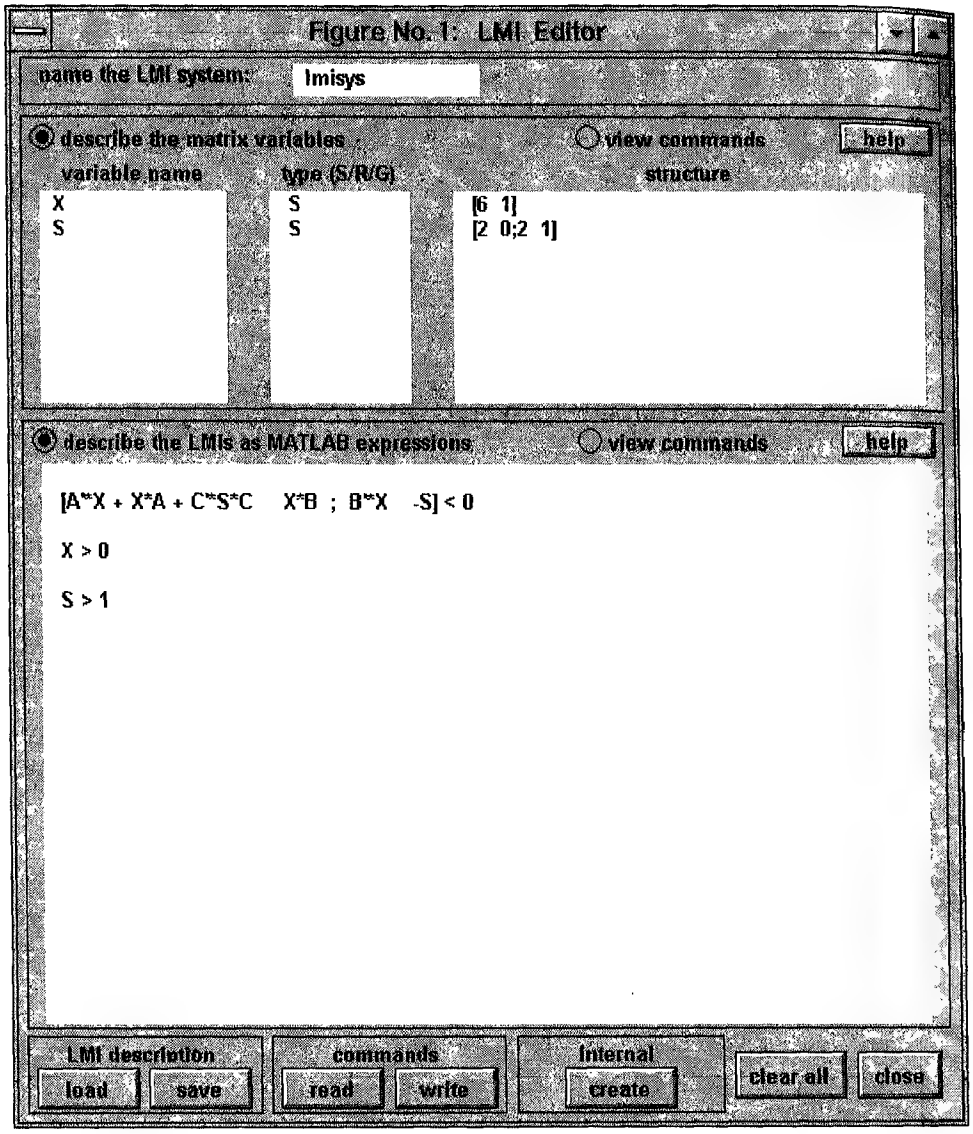


Figure 8-1: Graphical user interface `lmiedit`.

# Retrieving Information

Recall that the full description of an LMI system is stored as a single vector called the internal representation. The user should not attempt to read or retrieve information directly from this vector. Three functions called `lmiinfo`, `lminbr`, and `matnbr` are provided to extract and display all relevant information in a user-readable format.

## lmiinfo

`lmiinfo` is an interactive facility to retrieve qualitative information about LMI systems. This includes the number of LMIs, the number of matrix variables and their structure, the term content of each LMI block, etc. To invoke `lmiinfo`, enter

```
lmiinfo(LMISYS)
```

where `LMISYS` is the internal representation of the LMI system produced by `getlmis`.

## lminbr and matnbr

These two functions return the number of LMIs and the number of matrix variables in the system. To get the number of matrix variables, for instance, enter

```
matnbr(LMISYS)
```

# LMI Solvers

LMI solvers are provided for the following three generic optimization problems (here $x$ denotes the vector of decision variables, i.e., of the free entries of the matrix variables $X_1, \ldots, X_K$):

- **feasibility problem:**

  Find $x \in \mathbf{R}^N$ (or equivalently matrices $X_1, \ldots, X_K$ with prescribed structure) that satisfies the LMI system

  $$\mathcal{A}(x) < \mathcal{B}(x).$$

  The corresponding solver is called `feasp`.

- **minimization of a linear objective under LMI constraints**:

$$\text{Minimize } c^T x \text{ over } x \in \mathbf{R}^N \text{ subject to } \mathcal{A}(x) < \mathcal{B}(x).$$

  The corresponding solver is called `mincx`.

- **generalized eigenvalue minimization problem**:

$$\text{Minimize } \lambda \text{ over } x \in \mathbf{R}^N \text{ subject to}$$

$$\begin{aligned} \mathcal{C}(x) &< \mathcal{D}(x) \\ 0 &< \mathcal{B}(x) \\ \mathcal{A}(x) &< \lambda\, \mathcal{B}(x). \end{aligned}$$

  The corresponding solver is called `gevp`.

Note that $\mathcal{A}(x) < \mathcal{B}(x)$ above is a shorthand notation for general structured LMI systems with decision variables $x = (x_1, \ldots, x_N)$.

The three LMI solvers `feasp`, `mincx`, and `gevp` take as input the internal representation `LMISYS` of an LMI system and return a feasible or optimizing value $x^*$ of the decision variables. The corresponding values of the matrix variables $X_1, \ldots, X_K$ are derived from $x^*$ with the function `dec2mat`. These solvers are C-MEX implementations of the polynomial-time Projective Algorithm of Nesterov and Nemirovski [3, 2].

For generalized eigenvalue minimization problems, it is necessary to distinguish between the standard LMI constraints $\mathcal{C}(x) < \mathcal{D}(x)$ and the linear-fractional LMIs

$$\mathcal{A}(x) < \lambda\, \mathcal{B}(x)$$

attached to the minimization of the generalized eigenvalue $\lambda$. When using `gevp`, three rules should be followed to ensure proper specification of the problem:

- specify the LMIs involving $\lambda$ as $\mathcal{A}(x) < \mathcal{B}(x)$ (*without* the $\lambda$!)
- specify them *last* in the LMI system. `gevp` systematically assumes that the last $L$ LMIs are linear-fractional if $L$ is the number of LMIs involving $\lambda$
- add the constraint $0 < \mathcal{B}(x)$ or any other constraint that enforces it. This positivity constraint is required for well-posedness of the problem and is not automatically added by `gevp` (see the *Command Reference* chapter for details).

An initial guess `xinit` for $x$ can be supplied to `mincx` or `gevp`. Use `mat2dec` to derive `xinit` from given values of the matrix variables $X_1, \ldots, X_K$. Finally, various options are available to control the optimization process and the solver behavior. These options are described in detail in the *Command Reference* chapter.

The following example illustrates the usage of the `mincx` solver.

**Example 8.2:** Consider the optimization problem

$$\text{Minimize} \quad \text{Trace}(X) \quad \text{subject to} \quad A^T X + XA + XBB^T X + Q < 0 \qquad (8.9)$$

with data

$$A = \begin{pmatrix} -1 & -2 & 1 \\ 3 & 2 & 1 \\ 1 & -2 & -1 \end{pmatrix}; \quad B = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}; \quad Q = \begin{pmatrix} 1 & -1 & 0 \\ -1 & -3 & -12 \\ 0 & -12 & -36 \end{pmatrix}.$$

It can be shown that the minimizer $X^*$ is simply the stabilizing solution of the algebraic Riccati equation

$$A^T X + XA + XBB^T X + Q = 0.$$

This solution can be computed directly with the Riccati solver `care` and compared to the minimizer returned by `mincx`.

From an LMI optimization standpoint, problem (8.9) is equivalent to the following linear objective minimization problem:

$$\text{Minimize} \quad \text{Trace}(X) \quad \text{subject to} \quad \begin{pmatrix} A^T X + XA + Q & XB \\ B^T X & -I \end{pmatrix} < 0. \qquad (8.10)$$

Since $\text{Trace}(X)$ is a linear function of the entries of $X$, this problem falls within the scope of the `mincx` solver and can be numerically solved as follows:

1. Define the LMI constraint (8.9) by the sequence of commands

```
setlmis([])
X = lmivar(1,[3 1])     %  variable X, full symmetric

lmiterm([1 1 1 X],1,a,'s')
lmiterm([1 1 1 0],q)
lmiterm([1 2 2 0],-1)
lmiterm([1 2 1 X],b',1)

LMIs = getlmis
```

2. Write the objective Trace($X$) as $c^T x$ where $x$ is the vector of free entries of $X$. Since $c$ should select the diagonal entries of $X$, it is obtained as the decision vector corresponding to $X = I$, that is,

```
c = mat2dec(LMIs,eye(3))
```

Note that the function `defcx` provides a more systematic way of specifying such objectives (see p. 8-30 for details).

3. Call `mincx` to compute the minimizer `xopt` and the global minimum `copt = c'*xopt` of the objective:

```
options = [1e-5,0,0,0,0]
[copt,xopt] = mincx(LMIs,c,options)
```

Here `1e-5` specifies the desired relative accuracy on `copt`.

The following trace of the iterative optimization performed by `mincx` appears on the screen:

```
Solver for linear objective minimization under LMI constraints

Iterations   :    Best objective value so far

     1
     2                -8.511476
     3               -13.063640
***                 new lower bound:   -34.023978
     4               -15.768450
***                 new lower bound:   -25.005604
     5               -17.123012
***                 new lower bound:   -21.306781
     6               -17.882558
***                 new lower bound:   -19.819471
     7               -18.339853
***                 new lower bound:   -19.189417
     8               -18.552558
***                 new lower bound:   -18.919668
     9               -18.646811
***                 new lower bound:   -18.803708
    10               -18.687324
***                 new lower bound:   -18.753903
    11               -18.705715
***                 new lower bound:   -18.732574
    12               -18.712175
***                 new lower bound:   -18.723491
    13               -18.714880
***                 new lower bound:   -18.719624
    14               -18.716094
***                 new lower bound:   -18.717986
    15               -18.716509
```

```
***                 new lower bound:   -18.717297
    16                 -18.716695
***                 new lower bound:   -18.716873

 Result:  feasible solution of required accuracy
          best objective value:   -18.716695
          guaranteed relative accuracy:  9.50e-06
          f-radius saturation:  0.000% of R =  1.00e+09
```

The iteration number and the best value of $c^T x$ at the current iteration appear in the left and right columns, respectively. Note that no value is displayed at the first iteration, which means that a feasible $x$ satisfying the constraint (8.10) was found only at the second iteration. Lower bounds on the global minimum of $c^T x$ are sometimes detected as the optimization progresses. These lower bounds are reported by the message

```
***                 new lower bound:   xxx
```

Upon termination, `mincx` reports that the global minimum for the objective Trace(X) is $-18.716695$ with relative accuracy of at least $9.5 \times 10^{-6}$. This is the value `copt` returned by `mincx`.

4. `mincx` also returns the optimizing vector of decision variables `xopt`. The corresponding optimal value of the matrix variable $X$ is given by

```
Xopt = dec2mat(LMIs,xopt,X)
```

which returns

$$X_{\text{opt}} = \begin{pmatrix} -6.3542 & -5.8895 & 2.2046 \\ -5.8895 & -6.2855 & 2.2201 \\ 2.2046 & 2.2201 & -6.0771 \end{pmatrix}.$$

This result can be compared with the stabilizing Riccati solution computed by `care`:

```
Xst = care(a,b,q,-1)
norm(Xopt-Xst)

ans =
   6.5390e-05
```

# From Decision to Matrix Variables and Vice-Versa

While LMIs are specified in terms of their matrix variables $X_1, \ldots, X_K$, the LMI solvers optimize the vector $x$ of free scalar entries of these matrices, called the decision variables. The two functions `mat2dec` and `dec2mat` perform the conversion between these two descriptions of the problem variables.

Consider an LMI system with three matrix variables $X_1, X_2, X_3$. Given particular values `X1`, `X2`, `X3` of these variables, the corresponding value `xdec` of the vector of decision variables is returned by `mat2dec`:

```
xdec = mat2dec(LMISYS,X1,X2,X3)
```

An error is issued if the number of arguments following `LMISYS` differs from the number of matrix variables in the problem (see `matnbr`).

Conversely, given a value `xdec` of the vector of decision variables, the corresponding value of the $k$-th matrix is given by `dec2mat`. For instance, the value `X2` of the second matrix variable is extracted from `xdec` by

```
X2 = dec2mat(LMISYS,xdec,2)
```

The last argument indicates that the second matrix variable is requested. It could be set to the matrix variable identifier returned by `lmivar`.

The total numbers of matrix variables and decision variables are returned by `matnbr` and `decnbr`, respectively. In addition, the function `decinfo` provides precise information about the mapping between decision variables and matrix variable entries (see the *Command Reference* chapter).

# Validating Results

The LMI Lab offers two functions to analyze and validate the results of an LMI optimization. The function `evallmi` evaluates all variable terms in an LMI system for a given value of the vector of decision variables, for instance, the feasible or optimal vector returned by the LMI solvers. Once this evaluation is performed, the left- and right-hand sides of a particular LMI are returned by `showlmi`.

In the LMI problem considered in Example 8.2 on p. 8-19, you can verify that the minimizer `xopt` returned by `mincx` satisfies the LMI constraint (8.10) as follows:

```
evlmi = evallmi(LMIs,xopt)
[lhs,rhs] = showlmi(evlmi,1)
```

The first command evaluates the system for the value `xopt` of the decision variables, and the second command returns the left- and right-hand sides of the first (and only) LMI. The negative definiteness of this LMI is checked by

```
eig(lhs-rhs)

ans =
  -2.0387e-04
  -3.9333e-05
  -1.8917e-07
  -4.6680e+01
```

# Modifying a System of LMIs

Once specified, a system of LMIs can be modified in several ways with the functions `dellmi`, `delmvar`, and `setmvar`.

## dellmi

The first possibility is to remove an entire LMI from the system with `dellmi`. For instance, suppose that the LMI system of Example 8.1 on p. 8-6 is described in `LMISYS` and that we want to remove the positivity constraint on $X$. This is done by

```
NEWSYS = dellmi(LMISYS,2)
```

where the second argument specifies deletion of the 2nd LMI. The resulting system of two LMIs is returned in `NEWSYS`.

The LMI identifiers (*initial* ranking of the LMI in the LMI system) are not altered by deletions. As a result, the last LMI

$$S > I$$

remains known as the third LMI even though it now ranks second in the modified system. To avoid confusion, it is safer to refer to LMIs via the identifiers returned by `newlmi`. If `BRL`, `Xpos`, and `Slmi` are the identifiers attached to the three LMIs (8.6)-(8.8), `Slmi` keeps pointing to $S > I$ even after deleting the second LMI by

```
NEWSYS = dellmi(LMISYS,Xpos)
```

## delmvar

Another way of modifying an LMI system is to delete a matrix variable, that is, to remove all variable terms involving this matrix variable. This operation is performed by `delmvar`. For instance, consider the LMI

$$A^T X + XA + BW + W^T B^T + I < 0$$

with variables $X = X^T \in \mathbf{R}^{4\times 4}$ and $W \in \mathbf{R}^{2\times 4}$. This LMI is defined by

```
setlmis([])
X = lmivar(1,[4 1])      % X
W = lmivar(2,[2 4])      % W

lmiterm([1 1 1 X],1,A,'s')
lmiterm([1 1 1 W],B,1,'s')
lmiterm([1 1 1 0],1)

LMISYS = getlmis
```

To delete the variable $W$, type the command

```
NEWSYS = delmvar(LMISYS,W)
```

The resulting `NEWSYS` now describes the Lyapunov inequality

$$A^T X + XA + I < 0.$$

Note that `delmvar` automatically removes all LMIs that depended only on the deleted matrix variable.

The matrix variable identifiers are not affected by deletions and continue to point to the same matrix variable. For subsequent manipulations, it is therefore advisable to refer to the remaining variables through their identifier. Finally, note that deleting a matrix variable is equivalent to setting it to the zero matrix of the same dimensions with `setmvar`.

## setmvar

The function `setmvar` is used to set a matrix variable to some given value. As a result, this variable is removed from the problem and all terms involving it become constant terms. This is useful, for instance, to fix some variables and optimize with respect to the remaining ones.

Consider again Example 8.1 on p. 8-6 and suppose we want to know if the peak gain of $G$ itself is less than one, that is, if

$$\|G\|_\infty < 1 \ .$$

This amounts to setting the scaling matrix $D$ (or equivalently, $S = D^T D$) to a multiple of the identity matrix. Keeping in mind the constraint $S > I$, a legitimate choice is $S = 2 \times I$. To set $S$ to this value, enter

```
NEWSYS = setmvar(LMISYS,S,2)
```

The second argument is the variable identifier `S`, and the third argument is the value to which $S$ should be set. Here the value 2 is shorthand for $2 \times I$. The resulting system `NEWSYS` reads

$$\begin{pmatrix} A^T X + XA + 2C^T C & XB \\ B^T X & -2I \end{pmatrix} < 0$$
$$X > 0$$
$$2I > I \ .$$

Note that the last LMI is now free of variable and trivially satisfied. Hence it could be deleted altogether by

```
NEWSYS = dellmi(NEWSYS,3)
```

or

```
NEWSYS = dellmi(NEWSYS,Slmi)
```

if `Slmi` is the identifier returned by `newlmi`.

# Advanced Topics

This last section gives a few hints for making the most out of the LMI Lab. It is directed toward users who are comfortable with the basics described above.

## Structured Matrix Variables

Fairly complex matrix variable structures and interdependencies can be specified with `lmivar`. Recall that the symmetric block-diagonal or rectangular structures are covered by Types 1 and 2 of `lmivar` provided that the matrix variables are independent. To describe more complex structures or correlations between variables, you must use Type 3 and specify each entry of